Retrying failed back-end system operations with IBM Integration Bus

Dr. Hesham Soultan
Emad Mamoun
January 08, 2014

This article shows you how to develop a component to retry back-end system operations that has minimum impact on the performance of your main message flow. The simple, asynchronous component takes advantage of new IBM Integration Bus node capabilities.

Introduction

In many business integration projects, business rules require multiple retries of back-end system operations in case of failure. Such back-end operations are always transactional operations that create or update customer accounts or business items. You often need to implement a retry mechanism when rollback of the back-end operation is very difficult or even virtually impossible. The retry mechanism avoids the significant cost of manually processing the failed transactions, which otherwise may be necessary at the end of every business day.

As an example, consider a communications company whose customers pay monthly fees through a payment service company. If the payment service company sends a request to the communications company system, it is usually hard to roll it back or even make it a synchronous call, as the availability and performance of the back-end system would be degraded, especially during peak business hours. Therefore, if a request fails, the middleware must retry it a specified number of times before it reports a failure and sends it for manual processing by an employee at the end of the business day.

Asynchronous interaction

Asynchronous interaction between two components means that the sending component does not wait for a response to its request from the receiving component. Instead, the sending component sends the request and immediately continues with other activities. When the receiving component returns its response, the sending component is notified and processes the response whenever it needs to. In general, asynchronous interactions are preferred when applications are not time-critical, because the interacting systems do not spend any time waiting for responses, thus improving performance.
The best way to implement asynchronous interaction is through queued messaging. IBM® WebSphere® MQ provides highly efficient, enterprise-scale queued messaging, and its functionality is integrated into IBM Integration Bus.

Event timing in message flows
The IBM Integration Bus Toolkit provides the TimeoutControl and TimeoutNotification nodes to implement timed events. Other nodes such as the MQInput node also have an embedded timer. The configurable timer enables the node to perform timed browsing of queue contents.

The Retry message flow
The proposed retry message flow optimizes the two important parameters of a retry flow -- the method of interacting with the main flow, and the timing mechanism. The interaction method is asynchronous using WebSphere MQ messaging technology. The timing uses one of the existing nodes with an embedded timer, avoiding the overhead of adding one or more new timing nodes. Figure 1 shows the location of a retry flow in a typical integration flow. In case of failure in calling the back-end operation CreateUserPaymentWS, the retry flow is accessed through its input queue RETRY_QUEUE, outlined in red:

**Figure 1. Use case for retry message flow**

Here is an explanation of the retry mechanism:

1. In case of back-end call failure, the middleware integration flow, which belongs to the group of services that require call retrying, puts the failed request message in the input queue of the retry flow.
2. The messages in the input queue are browsed every n minutes, where n is a predefined value. This value affects the uncertainty of the retrying period of any request message. In other words, the period will be almost within the planned period, plus or minus n. Setting n to a high value leads to a high uncertainty. Setting it to a low value increases overhead, and may prevent the retry queue from processing all of the messages within the specified time period.
3. Messages in the retry input queue are retried after they have been in the queue for m minutes, where m is a predefined value. You define this period to give the back-end system time to resolve the problem that caused the failure. To determine whether to retry, the retry flow checks the browsed message creation time and compares it with the current time.
4. If the number of retries for a specific message exceeds a predefined limit, the flow stops retrying that message.
Figure 2 shows the retry message flow for a back-end system with MQ input. The normal flow path consists of MQInput, Filter, MQGet, Compute, and MQOutput nodes:

**Figure 2. Retry message flow**

The MQInput node listens to the retry queue, which holds the retry request messages that come from the middleware message flows. Configure the MQInput node using its internal timer to browse the messages in the queue every n minutes: Under Node properties, select **Advanced => Browse only**, and set the value of **Reset browse timeout**, as shown in Figure 3:

**Figure 3. MQInput node advanced properties**

The Filter node supports the timing activity by comparing the current time to the message creation time, and then deciding whether to process the message or leave it in the queue. If the comparison indicates that the retry time period has elapsed, the flow removes the message from the input queue and performs the retry by sending it to the back-end system. Otherwise, the flow browses the next message in the input queue. Listing 1 shows the ESQL code of the Filter node. Message creation time is set at the time it is received into the retry input queue.

**Listing 1. ESQL code of Filter node**

```esql
DECLARE waitTime INTEGER 1;
DECLARE creationHour INTEGER EXTRACT (HOUR FROM CAST (Root.Properties.CreationTime AS TIMESTAMP));
DECLARE currentHour INTEGER EXTRACT (HOUR FROM CURRENT_TIME);
DECLARE creationDay INTEGER EXTRACT (DAY FROM CAST (Root.Properties.CreationTime AS TIMESTAMP));
DECLARE currentDay INTEGER EXTRACT (DAY FROM CURRENT_DATE);
IF currentHour >= creationHour + waitTime OR currentDay > creationDay THEN
    RETURN TRUE;
ELSE
    RETURN FALSE;
END IF;
```
The role of the MQGet node is to consume the browsed message from the retry queue by getting it using its message ID. Configure the MQGet node to make it automatically get the message by its ID: Under Node properties, select Request => Get by message ID:

**Figure 4. Configuring MQGet node**

The Compute node RouteToBackend checks whether the number of times that the message has failed equals or exceeds a specified limit (such as 5). If so, then the flow routes it to the Failure queue (named DLQ in our example). Otherwise, the flow sends it to the back-end system queue to retry the operation. The ESQL code of the Compute node is shown in Listing 2. The target queue of the message is dynamically set by the code in OutputLocalEnvironment:

**Listing 2. The ESQL code of the compute node**

```esql
DECLARE discardTime INTEGER 5;
DECLARE retryCounter INTEGER COALESCE(InputRoot.MQRFH2.usr.RetryCounter, 0);

IF retryCounter > discardTime THEN
  SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName = 'FAILURE_QUEUE';
ELSE
  SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName = 'BackendRequestQueue';
  SET OutputRoot.MQRFH2.usr.RetryCounter = retryCounter + 1;
END IF;
```

The last node in the flow is the MQOutput node, whose name is set by the preceding Compute node. To support this dynamic queue name definition, under Node properties, select Advanced => Destination mode => Destination list, as shown in Figure 5:

**Figure 5. Advanced properties of MQOutput node**
Conclusion

This article showed you how to use IBM Integration Bus to retry back-end system operations from both business and technical perspectives. It described situations where retrying is important, then presented an example of a retry component that has demonstrated efficient performance in a production environment. The article also described the asynchronous component interaction method and its implementation using WebSphere MQ technology. Finally, the article described the retry timing mechanism based on the timer embedded in the IBM Integration Bus MQInput node.
Related topics

- **IBM Integration Bus resources**
  - IBM Integration Bus V9 documentation in IBM Knowledge Center
  - IBM Integration Bus product family page
  - Technical article: What's new in IBM Integration Bus V9
  - Announcement letter: IBM Integration Bus V9
  - IBM Knowledge Center topic: What's new in IBM Integration Bus for WebSphere ESB users
  - Download IBM Integration Bus Developer Edition
  - Follow IBM Integration Bus on Twitter
  - IBM Integration Bus forum
  - Track IBM Integration Bus user requirements

- **WebSphere MQ resources**
  - WebSphere MQ V7 documentation in IBM Knowledge Center
  - WebSphere MQ product page
  - WebSphere MQ product family
  - WebSphere MQ documentation library
  - IBM Redbook: WebSphere MQ V7 features and enhancements
  - Download a free trial version of WebSphere MQ
  - WebSphere MQ forum
  - WebSphere MQ SupportPacs

- **developerWorks resources**
  - developerWorks blogs
  - developerWorks events
  - IBM trials and downloads
  - developerWorks on Twitter

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)